EXHIBIT H

Deleted: 1
Deleted: 3
Deleted: 1
Deleted: 3
Deleted: 1
Deleted: 3
Deleted: 1
Deleted: 4
Deleted: 1
Deleted: 4
Deleted: 1
Deleted: 5
Deleted: 1
Deleted: 6
Deleted: 1
Deleted: 6
Deleted: 1
Deleted: 6
Deleted: 1
Deleted: 7
Deleted: 1
Deleted: 7
Deleted: 1
Deleted: 8
Deleted: 1
Deleted: 8
Deleted: 1
Deleted: 9
Deleted: 1
Deleted: 12
Deleted: 1
Deleted: 12
Deleted: 1
Deleted: 13
Deleted: 1
Deleted: 14
Deleted: 1
Deleted: 14
Deleted: 1
Deleted: 14
Deleted: 1
Deleted: 16
Deleted: 1
Deleted: 17
Deleted: 1
Deleted: 18
Deleted: 1
Deleted: 19
Deleted: 1
Deleted: 19
Deleted: 1
Deleted: 19

Deleted: 1
Deleted: 40
Deleted: 1
Deleted: 41
Deleted: 1
Deleted: 42
Deleted: 1
Deleted: 42
Deleted: 1
Deleted: 43
Deleted: 1
Deleted: 45
Deleted: 1
Deleted: 47
Deleted: 1
Deleted: 50
Deleted: 1
Deleted: 52
Deleted: 1
Deleted: 53
Deleted: 1
Deleted: 55
Deleted: 1
Deleted: 56
Deleted: 1
Deleted: 56
Deleted: 1
Deleted: 62
Deleted: 1
Deleted: 63
Deleted: 1
Deleted: 65
Deleted: 1
Deleted: 66
Deleted: 1
Deleted: 67
Deleted: 1
Deleted: 68
Deleted: 1
Deleted: 76
Deleted: 1
Deleted: 77

A WordML Primer

Peter Vogel

Introduction

This document, the WordML Primer, provides an easily approachable description of elements in the WordML schema that are important to document developers. The intended audience for this document includes application developers whose programs will read and write WordML documents. The text assumes that you have a basic understanding of XML 1.0, XML Namespaces, and the functionality of Microsoft Word. Each major section of the primer introduces new features of the language and describes those features in the context of concrete examples.

In this document you'll see how to:

- Create a document with typical Word structures: paragraphs, sections, and graphics
- Add typical document components including lists, tables, headers, footers, and title pages
- Format your documents by specifying formatting at any level within the document
- Define and use styles
- Insert fields into your document

Following this introduction to WordML is a reference to the WordML tags that are most useful to developers.

Structure of this document

After an initial overview of WordML and document-level properties and information, this whitepaper looks at WordML topics in the order that developers will, presumably, need them. This structure means that some elements are not discussed in detail in one location. For instance, the documentProperties element contains elements that affect how fields and headers are handled. As a result, different child elements of the documentProperties element are discussed in two different places in the document.

Section 1: An overview of WordML that describes the simplest possible WordML document and a summary of the top level WordML elements. Also covered in this section are the document information and properties section.
Section 2: Adding content to WordML document as unformatted text.
Section 3: Formatting text, including defining and using styles.
Section 4: Adding additional components to documents including lists, tables, headers and footers.
Section 5: Other topics: bookmarks, hyperlinks, fields
Section 6: The auxiliary tags added by Word to a WordML document to provide information about the document.

Section 1: WordML Overview

Top Level Elements, Namespace, Basic Document Structure

The top-level elements in a WordML document are

1) Document information (DocumentProperties element)
2) Font information (fonts element)
3) List style definitions (lists element)
4) Style definition (styles element)
5) Drawing defaults (shapeDefaults element)
6) DocSuppData (VBA code, toolbar customizations)
7) Document properties (docPr element)
8) The document's content (body element)

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

However, the simplest Word document consists of just five tags (and a single namespace). The five tags are:

wordDocument	the root element for a WordML document
body	the container for the displayable text
p	a paragraph
r	a run of identical WordML components with a consistent set of properties
t	a piece of text

The namespace for WordML is [redacted] and is normally associated with the WordML tags using a prefix of w. The simplest possible WordML document looks like this:

[redacted]

[redacted]

```
<?xml version='1.0'?>
<w:wordDocument
      xmlns:w='http://schemas.microsoft.com/office/word/2003/2/wordml' >

<w:body>
  <w:p>
      <w:r>
            <w:t>Hello, World.</w:t>
      </w:r>
  </w:p>
</w:body>

</w:wordDocument>
```

In Figure 1, you can see the resulting document, displayed in Microsoft Word.

**Insert graphic WordMLDev01.TIF
Figure 1: A WordML document in Microsoft Word

Tying the Document to Microsoft Word

If you want to save the document with the .XML extension the file will be treated as any other XML file by Windows. Double-clicking on the file, for instance, will open it in the standard XML processor (usually Internet Explorer). However, adding the mso-application processing instruction specifies Microsoft Word as the preferred application for processing the file. As a result, Word will open the XML document when the user double-clicks on the document's icon. This example shows the sample document with the mso-application tag added:

```
<?xml version='1.0'?>
<?mso-application progid='Word.Document'?>
<w:wordDocument
        xmlns:w='http://schemas.microsoft.com/office/word/2003/2/wordml' >
<w:body>
  <w:p>
        <w:r>
                <w:t>Hello, World.</w:t>
        </w:r>
  </w:p>
</w:body>

</w:wordDocument>
```

Section 2: Adding Text to the Document

The document's content is held in the body element. Text within the body element is kept in a nested set of three elements: t (a piece of text), r (a run of text within a paragraph), and p (a paragraph).

The t (text element), r (run), and p (paragraph) elements

The lowest level of this hierarchy is the t tag and is the container for the text that makes up the document's content. You can put as much text as you want in a t element—up to and including all of your document's content. However, in most WordML documents long runs of text will be broken up into paragraphs, strings with different formats, or be interrupted by line breaks, graphics, tables, and other components of a Word document.

[redacted]

Sections

In a WordML document, the layout of the page that your text appears in is controlled by the properties for that section of the document. However, there is no container element for sections in WordML. Instead, the information about a section is kept inside a sectPr (section properties) element that appears at the end of each section. Though a sectPr element isn't necessary in a WordML document, Word always inserts a sectPr tag at the end of any new document that it creates. This is a typical sectPr tag generated by Word when a document is created:

```
<w:sectPr>
        <w:pgSz w:w="12240" w:h="15840"/>
        <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
                w:header="720" w:footer="720" w:gutter="0"/>
        <w:cols w:space="720"/>
        <w:docGrid w:line-pitch="360"/>
</w:sectPr>
```

When new sections are added to a WordML document, the new sectPr elements must appear inside pPr elements (discussed later) inside p elements. This example show a sectPr element added to a document to mark the end of a section:

```
<w:p>
        <w:pPr>
                <w:sectPr>
                        <w:pgSz w:w="6120" w:h="7420" />
                        <w:pgMar w:top="720" w:right="720" w:bottom="720"
                                w:left="720" w:header="0" w:footer="0"
                                w:gutter="0" />
```

Deleted: u

[redacted]

Deleted: While sections are an important part of a Word document

Deleted: tag

Deleted: When

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

```
            </w:sectPr>
        </w:pPr>
</w:p>
```

Each sectPr tag marks the end of a section and the start of a new section. The child elements of the sectPr tag provide the definition of the section just ended. All the child elements for the sectPr tag are listed in Table 3.

While WordML does not have a container for sections, Word does generate sect tags that act as containers for sections. These are not part of WordML but belong to the auxiliary namespace ([redacted]). The sect tags (and other auxiliary tags) are discussed later in this document. [redacted]

Organizing Text

The following example has multiple t elements inside an r element (for the following examples, only the body tag and its children are shown):

Deleted: I'll

Deleted: show

```
<w:body>
  <w:p>
      <w:r>
            <w:t>Hello, World.</w:t>
            <w:t> How are you, today?</w:t>
      </w:r>
  </w:p>
</w:body>
```

While this document is valid, duplicating the t tag isn't necessary. This document would give the same result as the previous example:

```
<w:body>
  <w:p>
      <w:r>
            <w:t>Hello, World. How are you, today?</w:t>
      </w:r>
  </w:p>
</w:body>
```

Inserting Breaks

Typically, if you have multiple t tags in an r tag it's because you need to insert some other tag in between the pieces of text. In the following example, a br tag appears between the two t tags. The br tag will force the second t element to a new line when displayed in Word:

```
<w:body>
  <w:p>
      <w:r>
            <w:t>Hello, World. </w:t>
            <w:br w:type='text-wrapping'/>
            <w:t>How are you, today?</w:t>
        </w:r>
    </w:p>
</w:body>
```

The br tag's type property allows you to specify the kind of break ("page", "column", "text-wrapping"). The default is "text-wrapping" (a new line), so the val attribute in the previous example could have been omitted. Figure 2 shows the results of using a br tag between r elements.

**Insert graphic WordMLDev02.TIF
Figure 2: A Word document with a br tag between t tags

Creating Paragraphs

You use p elements to define new paragraphs (a br tag with text-wrapping is equivalent to the "soft break" in Word that's created by pressing Shift_Enter and doesn't start a new paragraph). A WordML document with text in two separate paragraphs would look like this:

```
<w:body>
    <w:p>
        <w:r>
            <w:t>Hello, World.</w:t>
        </w:r>
    </w:p>
    <w:p>
        <w:r>
            <w:t>How are you, today?</w:t>
        </w:r>
    </w:p>
</w:body>
```

The resulting document can be seen in Figure 3. As comparing Figures 2 and 3 shows, depending on your formatting options, the difference between using a br tag and p tags may not be visible. The display of a WordML document in Word may not reveal the underlying structure of the document.

Deleted: reflect

Deleted: real

**Insert graphic [redacted]
Figure 3: A Word document with multiple p tags

[redacted]

Tabs

The tab tag allows you to position text horizontally on a line. Tab tags move the following text to the next tab stop. Exactly where on the line that will be depends on how tabstops are defined in the document.

In this example, the text will appear on a single line but with each t tag's text positioned at a separate tabstop:

```
<w:p>
	<w:tab/>
	<w:r>
		<w:t>Hello, World.</w:t>
	</w:r>
	<w:tab/>
	<w:r>
		<w:t>How are you, today?</w:t>
	</w:r>
```

Tabstops are defined in the pPr element, which is also a child of the p tag. Within the pPr element you can set the tab stops for the paragraph by using tab tags with the tabs element. Three attributes on the tab element define the tabstop:

- val: type of tab
- pos: tab position from the left hand edge of the document, in twips
- leader: text to fill empty space between tabs

As an example, this paragraph has three tabstops at 1" (1,440 twips), 3" (4,320 twips), and 5" (7,200 twips), with each tabstop being a different type. In the example, the tab tags before the r tag move the text to the second tab stop:

```
<w:p>
	<w:pPr>
		<w:tabs>
			<w:tab  w:val ='center'   w:pos='1440'/>
			<w:tab  w:val='left'      w:pos='4320'/>
			<w:tab  w:val='decimal'   w:pos='7200'/>
		</w:tabs>
	</w:pPr>
	<tab/>
	<tab/>
	<w:r>
		<w:t>Hello, World.</w:t>
	</w:r>
</w:p>
```

Table 6 lists the attributes for the tab element and the options that you can use.

Graphics

Paragraphs aren't limited to text: A paragraph can also include a graphic. WordML stores graphics as a combination of Vector Markup Language (VML) and a binary representation of the image. A discussion of VML is outside the scope of this document, but this section shows how picture data fits into the structure of a WordML document.

Some shapes are very easy to add. For instance, to add a rectangle to your document, you only need the VML rectangle tag. The tag's style atrribute holds the information to draw a rectangle in the right place at the right size:

Deleted: To

[redacted]

Deleted: to use

Deleted: and tp provide

Deleted: enough information in the element's style property

Deleted: the

```
<v:rect id='_x0000_s1032' style='position:absolute;margin-left:63pt;margin-
top:4.2pt;width:54pt;height:45pt;z-index:1' />
```

To use the VML rect element you must add the VML namespace (urn:schemas-microsoft-com:vml) to the namespaces declared in your document. The office namespace may also be required if you intend to include anything more than the simplest autoshapes. Typically you'll establish these namespaces in the document's root element:

```
<w:wordDocument
        xmlns:w='http://schemas.microsoft.com/office/word/2003/2/wordml'
        xmlns:v='urn:schemas-microsoft-com:vml'
        xmlns:o='urn:schemas-microsoft-com:office:office'
        xml:space='preserve'>
```

Formatted: English (U.S.)

Your graphic must appear inside a pict element inside of a r element. This example adds a simple rectangle to the document using the VML rect tag:

```
<w:p>
        <w:r>
                <w:pict>
                        <v:rect id='_x0000_s1032'
                                style='position:absolute;margin-left:63pt;margin-
                        top:4.2pt;width:54pt;height:45pt;z-index:1' />
                 </w:pict>
          </w:r>
 </w:p>
```

Where a graphic consists of more than just a simple shape, you will also need to include a base64 encoded version of the graphic. Considerably more VML is required inside the pict tag:

```
<w:p>
        <w:r>
                <w:pict>
                        <v:shapetype id='_x0000_t75' coordsize='21600,21600'
                        o:spt='75' o:preferrelative='t'
                        path='m@4@5l@4@11@9@11@9@5xe'
                        filled='f' stroked='f'>
                                <v:stroke joinstyle='miter' />
                                <v:formulas>
                                        <v:f eqn='if lineDrawn pixelLineWidth 0' />
                                        <v:f eqn='sum @0 1 0' />
```

```
                    <v:f eqn='sum 0 0 @1' />
                    <v:f eqn='prod @2 1 2' />
                    <v:f eqn='prod @3 21600 pixelWidth' />
                    <v:f eqn='prod @3 21600 pixelHeight' />
                    <v:f eqn='sum @0 0 1' />
                    <v:f eqn='prod @6 1 2' />
                    <v:f eqn='prod @7 21600 pixelWidth' />
                    <v:f eqn='sum @8 21600 0' />
                    <v:f eqn='prod @7 21600 pixelHeight' />
                    <v:f eqn='sum @10 21600 0' />
                </v:formulas>
                <v:path o:extrusionok='f' gradientshapeok='t'
                    o:connecttype='rect' />
                <o:lock v:ext='edit' aspectratio='t' />
            </v:shapetype>
            <w:binData w:name='http://01000001.gif'>R0lGODlhQQAzAKI
            ...text removed...
            q18Ldi1baGzZt1/nZr07dW/Tv0cHDz3cc3HOxzMnt7x8
            </w:binData>
            <v:shape id='_x0000_i1025' type='#_x0000_t75'
                    style='width:48.75pt;height:38.25pt'>
                <v:imagedata src='http://01000001.gif'
                        o:title='FolderN' />
             </v:shape>
        </w:pict>
    </w:r>
</w:p>
```

Formatted: English (U.S.)

Section 3: Formatting Text

The most powerful tool discussed in this section is WordML styles. While it is possible to format your document by setting individual properties at the paragraph and run level it may not be your best choice. If you're doing more than setting bold, underline, or italics for a single run, using styles to format your document is a better choice for managing the appearance of your document.

Formatting Runs of Text

The rPr tag is a container that holds the property tags that define how a run is to be treated by Word. You can have multiple rPr tags within an r element. Table 1 lists all of the tags that can be included inside the rPr tag with their description ([redacted]).

[redacted]

Most of the run property tags have a single attribute called val that is limited to a specific set of values. For instance, the b (bold) tag, causes the text that follows it to be bolded when the b tag has its val attribute set to "on". In this example, both "Hello, World." and "How are you, today?" will be bolded because both sets of text are in the same run and follow the rPr tag with the b tag (The prefix (w:) on the val attribute is *not* optional):

```
<w:r>
        <w:rPr>
                <w:b w:val="on"/>
        </w:rPr>
        <w:t>Hello, World.</w:t>
        <w:br/>
        <w:t>How are you, today?</w:t>
</w:r>
```

Figure 4 shows the result of this change.

**Insert graphic WordMLDev04.TIF
Figure 4: Text in an r tag with the b property set.

If the val attribute isn't present for the b tag, it defaults to 'on'. So the tag <w:b/> is equivalent to the tag <w:b w:val='on'/>.

You can also use the b tag to suppress bolding like this:

```
<w:r>
        <w:rPr>
                <w:b />
        </w:rPr>
        <w:t>Hello, World.</w:t>
        <w:rPr>
                <w:b w:val="off"/>
        </w:rPr>
        <w:t>How are you, today?</w:t>
</w:r>
```

While most rPr tags use just the val attribute, there are exceptions (the asianLayout property, for instance, takes several attributes). Table 1 provides the values for the val attribute for each of the rPr properties, provided that the list of values is short. Where the element has multiple attributes, doesn't use the val attribute, or has a large number of values, the table gives the name of the type definition in the WordML schema that describes the element.

As an example, the underline element uses the val attribute but offers more choices than "on" and "off". This example gives the text a single, continuous underline (other options include "words", "double", "thick"):

```
<w:r>
        <w:rPr>
                <w:u w:val='single'/>
        </w:rPr>
        <w:t>How are you today?</w:t>
</w:r>
```

The result appears in figure 5.

**Insert graphic WordMLDev05.TIF
Figure 5: Applying the u property to text.

Formatting Paragraphs

The pPr element defines the properties for a paragraph. Table 2 lists the permitted child elements. As an example, within the pPr element the jc tag is used to control the paragraph's alignment. In this document, the text in the paragraph will be centered (see Figure 6):

```
<w:p>
        <w:pPr>
                <w:jc w:val='center'/>
        </w:pPr>
        <w:r>
                <w:t>Hello, World.</w:t>
        </w:r>
        <w:br/>
        <w:r>
                <w:t>How are you, today?</w:t>
        </w:r>
</w:p>
```

Deleted: ¶

**Insert graphic WordMLDev06.TIF

Figure 6: Centered text.

Styles

Styles allow you to create a group of style properties that can be applied as a unit either to individual paragraphs (within the pPr element) or runs (within the rPr element). Styles reduce the amount of WordML text that you have to produce and the amount of work require to make changes to your document's appearence. With styles, changing the appearance of all the pieces of text that share a common style has to be done in only one place: the style definition.

Using Styles

The pStyle element inside the pPr element specifies which style is to be use for all runs in the paragraph. In the [redacted]tags, the rStyle tag specifies the style for individual runs. The text inside the t tags will reflect a merging of the styles set at the pPr and set at the rPr level. There are no child elements in common between the pPr and rPr elements, so merging the two property sets is straightforward.

[redacted]

In this example

- The style "MyStyle" sets the paragraph-level properties
- "MyFirstRunStyle" sets the run-level properties for the text in the first t tag
- "MySecondRunStyle" sets the run-level properties for the text in the second t tag

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

```
<w:body>
        <w:p>
                <w:pPr>
                        <w:pStyle w:val='MyStyle'/>
                </w:pPr>
                <w:r>
                        <w:rPr>
                                <w:rStyle w:val='MyFirstRunStyle'/>
                        </w:rPr>
                        <w:t>Hello, World.</w:t>
                </w:r>
                <w:r>
                        <w:rPr>
                                <w:rStyle w:val='MySecondRunStyle'/>
                        </w:rPr>
                        <w:t>How are you, today?</w:t>
                </w:r>
        </ w:p>
</ w:body>
```

Defining Styles

[redacted]

Styles are defined in the WordML styles element, which is a top-level element under the wordDocument element. Within the styles tag, each style element defines a single style.

A style tag is a container element for tags that define the style (all the children for the style element are listed in Table 5).

The style tag itself takes three attributes: “type”, “styleId”, and “default”:

- The styleId attribute gives your style the name that you use to invoke the style in your WordML document
- The type attribute allows you to indicate what kind of style you’re defining: paragraph, character, table, or list. Styles used in the pPr tag must be paragraph styles; Styles in the rPr tag must be character styles
- You set the attribute called default to ‘on’ to indicate that this style is the default style for aparticular type of style: paragraph, character, table, list.

In the following example, three styles are defined:

- MyParagraphStyle, which is a paragraph style and flagged as the default style for paragraphs
- AnotherParagraph style, which is also a paragraph style
- EmphasisStyle, which is a character style

Deleted: the

```
<w:styles>
    <w:style w:type='paragraph' w:styleId='MyParagraphStyle' w:default='on'/>
    <w:style w:type='paragraph' w:styleId='AnotherParagraph'  w:default='off'/>
    <w:style w:type='character'  w:styleId='EmphasisStyle'    w:default='off'/>
</w:styles>
```

Deleted:

The following sample applies those styles. AnotherStyle is used for the first paragraph in the document. In the second paragraph, no paragraph style is specified, so the second paragraph will be formatted using the default style (MyParagraphStyle). However, within the r tag in the second paragraph, a character style is used to control the appearance of the text:

```
<w:body>
    <w:p>
        <w:pPr>
            <w:pStyle w:val='AnotherParagraph'/>
        </w:pPr>
        <w:r>
            <w:t>Hello, World.</w:t>
        </w:r>
    </w:p>
    <w:p>
        <w:r>
            <w:rPr>
                <w:rStyle w:val='Emphasis'/>
            </w:rPr>
            <w:t>How are you, today?</w:t>
        </w:r>
    </w:p>
</w:body>
```

Style Properties

You define a style by adding child elements to the style tags (all of the children are listed in Table 5). Within the style element, [redacted] elements allow you to define the formatting to be used at the r and p levels. The only limitation is that pPr elements used in a character style are ignored (and, as mentioned before, you can only refer to paragraph styles within an pPr element and only to character styles within an rPr element).

Putting it all together, this document defines a style that sets the justification for the paragraph (in the pPr element of the style) and combines bold and italics (in the rPr element of the style). The style is then used to format a paragraph.:

[redacted]

[redacted]

[redacted]

[redacted]

```
<w:styles>
        <w:style w:type='paragraph' w:styleId='ItalicBold' >
                [redacted]
                        <w:js  w:val='center'/>
                </w:pPr>
                [redacted]
                        <w:i   w:val='on'/>
                        <w:b  w:val='on'/>
                </w:rPr>
        </w:style>
</w:styles>
<w:body>
        <w:p>
                <w:pPr>
                        <w:pStyle  w:val='ItalicBold' />
                </w:pPr>
                <w:r>
                        <w:t>Hello, World.</w:t>
                </w:r>
        </w:p>
</w:body>
```

[redacted]

[redacted]

The result of applying this paragraph style using the pPr element inside the body is that the text will be italicized, bolded, and centered.

If you violate the restrictions that Word puts on using styles, Word won't raise an error but Word also won't apply your styles. Consider this example, which is similar to the previous example but has some key changes that prevent the style from being applied:

```
<w:styles>
        <w:style w:type='character' w:styleId='ItalicBold' >
                <w:pPr>
                        <w:js  w:val='center'/>
                </w:pPr>
                <w:rPr>
                        <w:i   w:val='on'/>
                <w:b  w:val='on'/>
            </w:rPr>
        </w:style>
</w:styles>
<w:body>
        <w:p>
            <w:pPr>
                <w:pStyle  w:val='ItalicBold' />
            </w:pPr>
            <w:r>
                <w:t>Hello, World.</w:t>
            </w:r>
        </w:p>
</w:body>
```

In this example, the ItalicBold style has its type attribute set to character. The result is that Word will ignore the use of the style in the pPr element inside the body.

In this example, the character version of the style is used correctly inside the rPr element but the result is still unfortunate:

```
<w:body>
        <w:p>
            <w:r>
                <w:rPr>
                    <w:rStyle  w:val='ItalicBold' />
                </w:rPr>
                <w:t>Hello, World.</w:t>
            </w:r>
        </w:p>
</w:body>
```

Because the style is specified as being a character style, the pPr element in the style definition (where the center justification is specified) will be ignored. The rPr element inside the style is applied, though. As a result, the text will be bolded and italicized but not centered.

You could also center, bold and italicized the text by making the ItalicBold style the default paragraph style and not specifying a style at the paragraph level:

Deleted: ¶
Y

Deleted: get the same effect by making

```
<w:styles>
        <w:style w:type='paragraph' w:styleId='ItalicBold'  default='on' >
            <w:pPr>
                <w:jc   w:val='center'/>
            </w:pPr>
            <w:rPr>
                <w:i   w:val='on'/>
                <w:b  w:val='on'/>
```

```
		</w:rPr>
	</w:style>
</w:styles>
<w:body>
	<w:p>
		<w:r>
			<w:t>Hello, World.</w:t>
		</w:r>
	</w:p>
</w:body>
```

Extending Styles

You can create a style by extending another style, using the basedOn tag. The basedOn tag allows you to define a single style and then create variations on that style by adding or overriding properties. This example defines an Italic style and then uses it as the base for a italicbold style:

```
<w:styles>
	<w:style w:type='paragraph' w:styleId='italic' >
		<w:rPr>
			<w:i  w:val='on'/>
		</w:rPr>
	</w:style>
	<w:style w:type='paragraph' w:styleId='italicbold' >
		<w:basedOn  w:val='italic'/>
		<w:rPr>
			<w:b  w:val='on'/>
		</w:rPr>
	</w:style>
</w:styles>
```

The order of the style elements with the styles element doesn't matter: a basedOn style can extend style elements that precede or follow it.

Other useful child elements of the style tag include:

- name: establishes the name to be displayed in the style drop down list.
- locked: prevents users from redefining the style
- hidden: prevents users from seeing a style
- next: specifies the style to be used on a new paragraph created when the user presses the Enter key at the end of the current paragraph.

As a more comprehensive example, the following style establishes

- A style called ReferenceName
- In Word, the user will see the style called DisplayName in the Style drop down list
- The style is locked to prevent it being redefined by the user
- When the user presses the enter key, the next paragraph will be in the italicbold style

- [redacted]

Deleted: While I've used

Deleted: I've set

Deleted: '

[redacted]

```
<w:style w:type='paragraph' w:styleId='ReferenceName' >
        <w:name w:val='DisplayName' />
        <w:locked w:val='on' />
        <w:hidden w:val='off'/>
        <w:next  w:val='italicbold'/>
        <w:rPr>
                <w:i   w:val='on'/>
        </w:rPr>
</w:style>
```

This paragraph uses the style just defined, reference with the name specified in the style's styleId attribute:

```
<w:p>
        <w:pPr>
                <w:pStyle w:val='ReferenceName'/>
        </w:pPr>
        <w:r>
                <w:t>Hello, World</w:t>
        </w:r>
</w:p>
```

Figure 7 shows the style applied to the first paragraph in the document. In Word's toolbar, the Style drop down list shows the name established for the style through the name element in the style tag. The second paragraph in Figure 7 was created by pressing the enter key at the end of the first paragraph and is in the italicbold style.

**Insert graphic WordMLDev07.TIF
Figure 7. The DisplayName style in use

Property Conflicts

Because properties can be set at the style, p, and r levels, Word must deal with conflicts between the three levels. In this example, for instance, Word must reconcile

- The bold setting in the rPr element before the text
- Any bold setting that might be made in the character style, MyFirstRunStyle
- Any bold setting made in the paragraph style, MyStyle

Formatted: Bulleted + Level: 1 + Aligned at: 0.75" + Tab after: 1" + Indent at: 1"

Finally, Word must deal with any bold setting made in the default character or paragraph styles:

Formatted: Indent: Left: 0.25"

```
<w:body>
        <w:p>
                <w:pPr>
                <w:pStyle w:val="MyStyle"/>
          <w:pPr>
          <w:r>
                <w:rPr>
                      <w:rStyle w:val='MyFirstRunStyle'/>
                      <w:b w:val='on'/>
                </w:rPr>
                <w:t>Hello, World.</w:t>
          </w:r>
    </ w:p>
</ w:body>
```

There are three rules that are used to reconcile settings for properties that are either 'on' or 'off':

1. The local setting controls the text: settings made in the rPr or pPr elements for the run override any settings made in a style.
2. When styles are applied using the pStyle or rStyle tags, if either of the styles turn on the feature, then the feature is turned on.
3. The setting in the default style is only relevant if no explicit style is applied to the text and no local setting is made.

Applying the rules to the previous example, the text "Hello, World" will be bolded because of the b tag in the rPr element of the run. If the b tag hadn't been used in the rPr element then if either of MyStyle or MyFirstRunStyle turned on bolding, then the text would be bolded—even if one of the styles turned bolding off. Finally, any bold setting in a default paragraph or character style would apply only if no style was applied to the text.

Formatted: Indent: Left: 0.5", Hanging: 0.25", Tabs: 0.75", List tab + Not at 1.17"

Formatted: Bullets and Numbering

Formatted: Indent: First line: 0.5", No bullets or numbering

[redacted]

[redacted]

WordML provides two separate kinds of support for fonts:

1) Providing Word with information about the fonts used in the document
2) Controlling the font used to display [redacted] in the document.

Defining Fonts

You can define the fonts used in your document using the fonts tag. For each font that you use, you can specify a variety of properties that allow Word to manage the document's fonts and make appropriate substitutions when the requested font isn't available. Setting these properties requires an understanding of font information that's beyond this document. The font tag has no direct effect on your document's appearance but is simply a place to supply Word with information about the fonts used by the document.

Using Fonts

Within the fonts tag, [redacted] element specifies the default fonts for the document. This tag directly controls which font is to be used to display the text in the document (unless overridden by a style or an rPr child element).The defaultFont tag has a set of attributes that let you specify the default fonts for four character sets: ascii, fareast,

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

h-ansi, and cs (complex scripts). This is one tag that controls what font is to be used in displaying the document.

You can override the default font by using the rFonts tag in the rPr element. This can be done either in the rPr element preceeding the t tag with the text, in an rPr element inside a pPr element, or in a style. The rFonts tag takes the same attributes as the defaultFonts tag. As an example, this tag sets the font for a run to Tahoma:

Deleted: To apply a font to text to format your document, y

Deleted: use

Deleted: for the run

[redacted]

Deleted: 'Arial'

Deleted: 'Arial'

Deleted: 'Arial'

[redacted]

[redacted]

[redacted]

[redacted]

Formatting the Section

At the section level, formatting information is held with the [redacted] of the section. Within the sectPr element, child elements allow you to control the page's size, the page margins, and to define columns for the page.

[redacted]

Setting Page Size and Margins

In the sectPr tags, there are two tags that control your page layout:

- pgSz: page size. The attributes on this tag let you set the height and width of your page
- pgMar: page margin. The attributes on this tag let you set the width of the left, right, top, and bottom margins.

This pgSz tag uses the w attribute to set a page width of 12,240 twips (8.5 inches) and the h attribute to set the height at 15,840 twips (11 inches):

```
<w:pgSz w:w='12240' w:h='15840' [redacted]/>
```

[redacted]

The following pgMar tag sets the top and bottom margins at 1,440 twips (1 inch) and the left and right margins at 1,800 twips (1.25 inches). In addition, the header and footer have 720 twips (0.5 inches) set aside for them:

```
<w:pgMar w:top='1440' w:right='1800' w:bottom='1440' w:left='1800'
                  w:header='720' w:footer='720' />
```

The pgMar tag also lets you specify how much space is to be set aside for the gutter. When pages are bound together, some part of the page is lost to the binding process. In the previous example, no space has been left for the gutter. This next example sets aside a quarter inch for the gutter:

Deleted: this

```
<w:pgMar w:top='1440' w:right='1800' w:bottom='1440' w:left='1800'
                        w:header='720' w:footer='720' w:gutter='360'/>
```

Normally documents are bound down their inside edges. If your documents are bound along the top, you'll need to specify that in the docPr tag:

```
<w:docPr>
        <w:gutterAtTop w:val='on'/>
</w:docPr>
```

To force the gutter to the right-hand side of the page, use the rtlGutter property.

Columns

Columns are also defined in the sectPr tag using the cols tag. If your columns are all the same width, then you only need to specify the number of columns (in the num attribute) and the space between columns (in the space attribute):

```
<w:cols w:num='4' w:space='720'/>
```

If the columns are unequal width, then you must insert col tags inside the cols element. You must still specify the number of columns on the cols element. You must also specify that you are turning off equalWidth, using the equalWidth attribute:

Deleted: that

```
<w:cols w:num='4' w:sep='on' w:space='1440' w:equalWidth='off'>
```

For each column tag, you specify the width of the column and the space following it. For the last column, you don't have to specify the space following:

```
<w:cols w:num='4' w:sep='on' w:space='1440' w:equalWidth='off'>
        <w:col w:w='1440' w:space='500'/>
        <w:col w:w='2880' w:space='500'/>
        <w:col w:w='1080' w:space='750'/>
        <w:col w:w='1080'/>
</w:cols>
```

The good news is that's all you have to do. Word will take care of snaking the content of the t tags in your body through the columns.

Section 4: Document Components

This section shows how to add lists, tables, headers, footers, and title page elements to a WordML document. You'll also see how to add both document properties and Office information to your document.

Lists

In WordML, lists are a series of paragraphs that have a listing style applied to them, with each item in the list in a separate paragraph. What distinguishes a "list paragraph" from an "ordinary paragraph" is the presence of a listPr tag in the pPr tag in the paragraph. The listPr tag specifies the listing style to be used with the paragraph's content and the level of the list. Here is a sample of a list with two items; two paragraphs with listPr elements:

```
<w:p>
        <w:pPr>
                <w:listPr>
                        <w:ilvl w:val="0" />
                        <w:ilfo w:val="1" />
                </w:listPr>
        </w:pPr>
        <w:r>
                <w:t>item 1</w:t>
        </w:r>
</w:p>
<w:p>
        <w:pPr>
                <w:listPr>
                        <w:ilvl w:val="0" />
                        <w:ilfo w:val="1" />
                </w:listPr>
        </w:pPr>
        <w:r>
                <w:t>Item 2</w:t>
        </w:r>
</w:p>
```

Formatted: English (U.S.)

Formatted: English (U.S.)

Formatted: English (U.S.)

Only two elments can appear inside the listPr tag:

- ilvl: The list's level number. This number is incremented as lists are nested within lists.
- ilfo: The list style to use. This number must refer to a list style defined in the lists section of the document

As an example of the ilvl in action, consider the following list:

1. Types of Websites
    a. Applications
    b. Content

    c. Hybrid
2. WayFinding
    a. Planning strategies
    b. Executing plans with feedback

There are actually three lists in the example. First, there is an outer list with two items ("Types of Websites" and "WayFinding", numbered 1 and 2). Within those items are two nested lists. The first is the list consisting of "Applications", "Content", and "Hybrid"; the second list consists of "Planning strategies" and "Executing plans with feedback". In WordML, this example will consist of seven paragraphs, one for each list item. The paragraphs in different lists are at different paragraphs levels and have different listing styles assigned to them.

For the first three paragraphs, the WordML would look like this:

```
<w:p>
        <w:pPr>
                <w:listPr>
                        <w:ilvl w:val="0" />
                        <w:ilfo w:val="2" />
                </w:listPr>
        </w:pPr>
        <w:r>
                <w:t>Types of Web sites</w:t>
        </w:r>
</w:p>
<w:p>
        <w:pPr>
                <w:listPr>
                        <w:ilvl w:val="1" />
                        <w:ilfo w:val="2" />
                </w:listPr>
        </w:pPr>                                                    Formatted: French (France)
        <w:r>
                <w:t>Applications</w:t>
        </w:r>                                                      Formatted: English (U.S.)
</w:p>
<w:p>
        <w:pPr>
                <w:listPr>
                        <w:ilvl w:val="1" />
                        <w:ilfo w:val="2" />
                </w:listPr>
        </w:pPr>                                                    Formatted: French (France)
        <w:r>
                <w:t>Content</w:t>
        </w:r>                                                      Formatted: English (U.S.)
```

```
</w:p>
```

The entry in the outermost list ("Types of Websites") has an ilvl of "0". The next paragraph, which is the first item of the nested list ("Applications") has an ilvl of "1", indicating that the paragraph is nested one level deep. All of the paragraphs use the same listing style, specified in the ilfo tag as "2".

The value in the ilfo tag refers to a list element which appears inside the lists element before the body element. The list element, in turn, associates the ilfo id with a particular list definition. The following list tag, for instance, defines list "1" as using list definition "2":

```
<w:lists>
	<w:list w:ilfo="1">
		<w:ilst w:val="2" />
	</w:list>
</w:lists>
```

The list element can contain one other element, lvlOverride. The lvlOverride element is a container containing elements that override settings in the list definition. These overrides can include a new starting number for the list and formatting. By using lvlOverride you can specify settings for one particular list (or part of a list) without having to create a whole new list definition.

The actual list definitions are defined inside the listDef element, which also appears inside the lists element. On the listDef element, the [redacted] (which must be numeric) specifies the list name that is used in the ilfo tag of the list element. All of the children of the listDef element are given in Table 15.

[redacted]

Within the listDef element, lvl elements define how list items at each level are to be formatted. The format information inside a lvl tag can include a pPr element (containing formatting for p tags) and an rPr element (containing formatting for r tags), among other elements. The pPr and rPr settings will automatically be applied to the p and r tags that make up the list item's paragraph.

Also within the lvl element, the start element specifies the starting number for the list.

Here is a sample listDef definition that defines two levels of a list (Word typically generates eight levels of definition for a listDef):

```
<w:listDef w:listDefId="0">
	<w:lvl w:ilvl="0">
		<w:start w:val="1" />
		<w:lvlText w:val="%1." />
		<w:lvlJc w:val="left" />
		<w:pPr>
			<w:tabs>
				<w:tab w:val="list" w:pos="1080" />
			</w:tabs>
			<w:ind w:left="1080" w:hanging="720" />
```

```
                </w:pPr>
                <w:rPr>
                        <w:rFonts w:hint="default" />
                </w:rPr>
        </w:lvl>
        <w:lvl w:ilvl="1" w:tplc="56325532">
                <w:start w:val="1" />
                <w:nfc w:val="4" />
                <w:lvlText w:val="%2." />
                <w:lvlJc w:val="left" />
                <w:pPr>
                        <w:tabs>
                                <w:tab w:val="list" w:pos="1800" />
                        </w:tabs>
                        <w:ind w:left="1800" w:hanging="720" />
                </w:pPr>
                <w:rPr>
                        <w:rFonts w:hint="default" />
                </w:rPr>
        </w:lvl>
</listDef>
```

Formatted: English (U.S.)

Formatted: English (U.S.)

Headers, Footers, and Title Pages

WordML lets you add headers, footers and a title page to your document. In WordML, headers and footers are just another kind of paragraph.

Headers and footers are defined in the sectPr tag that marks the end of the section. In the sectPr tag, the hdr element contains the definition of the header for the section; the ftr tag contains the definition for the footer. Within the hdr and ftr tags, the content of the tag is treated like the content of the body tag: p, r, and t tags are used to hold the text that make up the header or footer.

Here's an example of the definition of a header and a footer:

```
<w:sectPr>
        <w:hdr w:type='odd'
                <w:p>
                        <w:pPr>
                                <w:pStyle w:val='Header'/>
                        </w:pPr>
                        <w:r>
                                <w:t>My Header</w:t>
                        </w:r>
                </w:p>
        </w:hdr>
        <w:ftr w:type='odd'
                <w:p>
                        <w:pPr>
                        <w:pStyle w:val='Footer'/>
                </w:pPr>
                <w:r>
                        <w:t>My Footer</w:t>
                </w:r>
        </w:p>
</w:ftr>
```

You can use any style that you want to control the formatting of a header or fooer. A typical header style might look like this:

```
<w:style w:type='paragraph' w:styleId='Header' >
        <w:name w:val='header'/>
        <w:basedOn w:val='Normal'/>
        <w:pPr>
                <w:pStyle w:val='Header'/>
                <w:tabs>
                        <w:tab w:val='center' w:pos='4320'/>
                        <w:tab w:val='right' w:pos='8640'/>
                </w:tabs>
        </w:pPr>
</w:style>
```

The hdr and ftr elements have a type attribute that takes one of three values: "even", "odd", and "first". If you're only using one hdr or ftr then the type attribute must be set to "odd".

To have a different header (or footer) on even and odd pages you will need two hdr elements, one with its type attribute set to "even" and the other hdr element with its type attribute set to "odd":

```
<w:sectPr>
        <w:hdr w:type='odd'
                <w:p>
                        <w:pPr>
                                <w:pStyle w:val='Header'/>
                        </w:pPr>
                        <w:r>
                                <w:t>My Odd Header</w:t>
                        </w:r>
                </w:p>
        </w:hdr>
        <w:hdr w:type='even'
                <w:p>
                        <w:pPr>
                                <w:pStyle w:val='Header'/>
                        </w:pPr>
                <w:r>
                        <w:t>My Even Header</w:t>
                </w:r>
        </w:p>
</w:hdr>
```

You must also add the evenAndOddHeaders tag to the docPr tag at the top of the document:

```
<w:docPr>
        <w: evenAndOddHeaders/>
</w:docPr>
```

If you set the type attribute of a hdr or ftr to "first" then the hdr or ftr will only be used on the first page (even if it's the only hdr/ftr in the document). You don't have to add any tags to the document properties to use this option, but you do need to add the titlepg tag to the end of the sectPr element, following the definition of your headers and footers:

```
<w:sectPr>
        <w:hdr w:type='even'
                <w:p>
                        <w:pPr>
                                <w:pStyle w:val='Header'/>
                        </w:pPr>
                        <w:r>
                                <w:t>My Title Page Header</w:t>
                        </w:r>
                </w:p>
        </w:hdr>
        <w:titlePg/>
</w:sectPr>
```

To ensure that your headers and footers display correctly, you should allocate the space on the page to display them. For this, you'll need to control your page's layout, as described in formatting your section.

Tables

In WordML, tables are defined with the tbl tag (Table 9 lists the high level table tags). Within the tbl tag,

- the tblPr tag contains property settings for the table (Table 10)
    - Wthin the tblPr element, the tblPPr tag holds settings that control the table's position (Table 11)
- the tblGrid element contains gridCol elements that defines the widths of the columns in the table
- tr tags define the rows for the table

- Within the tr element, trPr tags define properties for the row (Table 12)
- Also within the tr tags, tc tags define cells within the row
  - Within the tc tags, tcPr tags define properties for the cell (Table 13)
  - Also within the tc tags is the table's content

The content of a tc element can be one or more p tags or even another table.

The following example will define a table with two columns and a single row. The opening tbl tag is followed by a tblPr element containing a set of table properties. As is typical in WordML, each property is an empty element with a single attribute called val that contains the value for the property. In this example

- the tblStyle element specifies that the table is to use the "[redacted] style (which would have to be defined in the styles element at the top of the document)
- The tblW element specifies the total width of the table (in this case "auto")
- The tblLook element contains a bitmask that specifies how rows are to be formatted. The 1E0 is the result of adding up the four options: header row formatting, last row formatting, header column formatting, and last column formatting

[redacted]

Deleted: , as

```
<w:tbl>
	<w:tblPr>
		<w:tblStyle w:val="TableGrid"/>
		<w:tblW w:w="0" w:type="auto"/>
		<w:tblLook w:val="000001E0"/>
	</w:tblPr>
```

The next element inside the tbl tag is the tblGrid which contains one gridCol element for each column in the table. The w attribute on the gridCol element gives the width of the column (in twips). In this example, there are two columns, one 1770 twips and one 1400 twips wide:

```
	<w:tblGrid>
		<w:gridCol w:w="1770"/>
		<w:gridCol w:w="1400"/>
	</w:tblGrid>
```

With the table now defined, tr elements contain the cells with the table's content. The tr element can contain a trPr element holding properties for the row (e.g. the row height and whether it can be split across a page). The following example omits the trPr element.

Within the tr element the row's cells, which are defined by tc tags, contain the table's content. Within tc tags, a tcPr element contains the properties for the cell. In the following example

- the tcW element's w attribute specifies that the cell is 1770 units wide
- the type attribute specifies that the unit is twips

Also within the tc element is the cell's content. In this example, the content is a p tag with a single run with a single piece of text:

```
        <w:tr>
                <w:tc>
                        <w:tcPr>
                                <w:tcW w:w="1770" w:type="dxa"/>
                        </w:tcPr>
                        <w:p><w:r><w:t>Hello, World</w:t></w:r></w:p>
                </w:tc>
```

Cells can be merged by using the Vmerge and Hmerge elements in the tcPr element. An empty Vmerge or Hmerge element with its restart attribute set to "restart" marks the start of a merged range. A Vmerge or Hmerge element without any attributes marks the end of the merged cells. Cells between the first and last cell must have a Vmerge or Hmerge element with the val attribute set to "continue". In this example, the last cell in the first row starts a merge that is continued with the cell below it:

Deleted: '

Deleted: each

Deleted:

Deleted: will be merged with the cell below it:

```
        <w:tr>
                <w:tc>
                        <w:p><w:r><w:t>First cell, first row</w:t></w:r></w:p>
                </w:tc>
                <w:tc>
                        <w:tcPr>
                                <w:vmerge w:val="restart"/>
                        </w:tcPr>
                        <w:p><w:r><w:t>Last cell, first row </w:t></w:r></w:p>
                </w:tc>
        </w:tr>
        <w:tr>
                <w:tc>
                        <w:p><w:r><w:t>First cell, second row</w:t></w:r></w:p>
                </w:tc>
                <w:tc>
                        <w:tcPr>
                                <w:vmerge />
                        </w:tcPr>
                        <w:p><w:r><w:t>Last cell, second row </w:t></w:r></w:p>
                </w:tc>
        </w:tr
```

Deleted: V

Deleted: V

[redacted]

The graphic in Figure 8 shows the results of merging the two cells. The space that the second cell would occupy is now just a continuation of the cell above it and can have no separate content. The content specified in the WordML file for the last row, second cell, disappears when displayed in Word. The content is still present, though, and can be retrieved through Word's object model.

**Insert Graphic WordMLDev13.tif

Figure 8. The results of merging two cells.

Table 10 lists the table property elements; table 11 lists the table positioning elements; table 12 lists the child elements for table row properties; table 13 lists the child elements for table cell properties.

Document Properties

The document has a set of properties, held in the docPr element. Table 8 lists the properties that can be set at the document level. Some useful settings for developers creating documents include:

- view: Controls the view mode in Word. The val attribute on this element can be set to "none", "print", "outline", "master-pages", "normal", and "web". The percent attribute can be used to set the percentage of the zoom.
- zoom: Controls how large or small the document appears on the screen in Word. The val attribute on this element can be set to "none", "full-page", "best-fit", "text-fit".
- documentProtection: Controls whether readers can make unintentional changes to all or part of an online form or document.
- footnotePr: This container element holds the property settings and content for all footnotes in the document
- endnotePr: The container element holds the property settings and content for endnotes

As an example, the following example shows a set of document properties that set the user's view to normal, zooms the view to full page, and prevents the user from changing the formatting in the document:

Deleted: s

```
<w:docPr>
        <w:view  w:val='normal'/>
        <w:zoom w:val='full-page' w:per-cent='100'/>
        <w:documentProtection w:formatting='on' w:enforcement='on'/>
</w:docPr>
```

Document Information

The DocumentProperties element performs a different function from the docPr element. Like docPr, the DocumentProperties is a container for other elements. The DocumentProperties tag, however, is not part of the WordML namespace but is part of the [redacted] namespace, a set of tags common to all Office applications.

[redacted]

The DocumentProperties elements contain meta-information about the document, including the document's title, version, and author. Some statistics about the document are also kept in the DocumentProperties, including the number of characters, pages, lines, and paragraphs. Here's a sample DocumentProperties element:

```
<o:DocumentProperties>
        <o:Title>Sample Document</o:Title>
        <o:Author>Peter Vogel</o:Author>
        <o:Pages>1</o:Pages>
        <o:Words>2</o:Words>
        <o:Characters>15</o:Characters>
        <o:Lines>1</o:Lines>
        <o:Paragraphs>1</o:Paragraphs>
        <o:Version>11.4920</o:Version>
</o:DocumentProperties>
```

Section 5: Other Topics

Bookmarks

Bookmarks are not part of the WordML namespace but are part of the [redacted] (conventionally prefixed with '[redacted]'). In a WordML document, annotation tags, which are empty elements, bracket the area that is bookmarked. An annotation tag with its type attribute set to Word.Bookmark.Start marks the start of a bookmark area; an annotation tag with its type set to Word.Bookmark.End marks the end of the bookmark.

[redacted]

[redacted]

In this example, a complete paragraph (containing the text "Inside bookmark") has been bookmarked with a book mark called "MyBookmark":

```
<w:p><w:r><w:t>Before bookmark</w:t> </w:r> </w:p>
<aml:annotation aml:id='0' w:type='Word.Bookmark.Start'
                                        w:name='MyBookmark' />
<w:p><w:r><w:t>Inside bookmark</w:t></w:r></w:p>
<aml:annotation aml:id='0' w:type='Word.Bookmark.End' />
<w:p><w:r><w:t>After bookmark</w:t></w:r></w:p>
```

Where a bookmark is inserted without enclosing any text, the annotation tags will be inserted between r tags:

```
<w:p><w:r><w:t>text sur</w:t></w:r>
<aml:annotation aml:id='1' w:type='Word.Bookmark.Start'
                                  w:name='MyOtherBookmark' />
<aml:annotation aml:id='1' w:type='Word.Bookmark.End' />
<w:r><w:t>rounding bookmark</w:t></w:r></w:p>
```

In addition to the type attribute, which identifies an annotation tag as being used as a bookmark, two attributes of the annotation tag are used in working with bookmarks:

- The name attribute holds the name of the bookmark and allows you to identify the bookmark
- The id attribute's value is what links the end tag for a bookmark to its start tag. In the previous examples, for instance, the id attribute has identical values for the Word.Bookmark.End element and the Word.Bookmark.Start elements.

Fields

You can define fields in your document with WordML. A typical Word document with several form fields can be seen in Figure 9.

[redacted]

[redacted]

**Insert graphic [redacted]

Figure 9: A Word document with several fields to enter.

[redacted]

A field is, effectively, a kind of declarative programming. A field is a set of instructions on how part of the document is to be processed. Also included in the field definition are any input parameters and the results of the processing. WordML supports two kinds of fields:

Simple fields: Fields whose instructions are simple text

Complex fields: Fields whose instructions can include more than just simple text (e.g. references to other fields, rich text).

Simple Fields

Simple fields are defined with the fldSimple tag. The fldSimple tag has an instr (instruction) attribute whose contents define the field's behavior. Within the fldSimple element, an r element holds the results of processing the instructions. For instance, this example creates a simple field that will [redacted] [redacted] into the text:

```
<w:fldSimple w:instr="AUTHOR \* Upper \* MERGEFORMAT">
    <w:r>
        <w:t>Peter Vogel</w:t>
    </w:r>
</w:fldSimple>
```

[redacted]

Formatted: German (Germany)

Complex Fields

Complex fields appear in WordML as a series of r tags inside a paragraph. Each r tag contains one part of the field's definition. Three r elements contain fldChar elements which mark the three parts of a complex field definition:

- the beginning of the field definition
- the end of the field instructions
- the end of the field definition

The fldChar tag is used to mark each of these three parts. The fldCharType attribute on the fldChar tag is set to "begin", "separate", and "end" to mark the parts of the field definition. The field instructions are placed in the instrText elements. The instrText elements appear between the r element that marks the beginning of the field definition and the r element that marks the end of the instructions. The results of the field's processing are placed between the r element that marks the end of the instructions and the r relement that marks the end of the field definition.

In order to make it easier to find the form field when processing the document, a bookmark can be added to identify the field.

One kind of complex field is a form text field A set of WordML tags to create a single form field inside a p tag would look like this:

```
<w:p>
    <w:r>
        <w:fldChar w:fldCharType="begin"/>
    </w:r>
    <w:r>
        <w:instrText>FORMTEXT</w:instrText>
    </w:r>
```

```
        <w:r>
                <w:fldChar w:fldCharType="separate" />
        </w:r>
        <w:r>
                <aml:annotation aml:id="0" w:type="Word.Bookmark.Start"
                        w:name="MyField" />
                <w:t>     </w:t>
                <aml:annotation aml:id="0" w:type="Word.Bookmark.End" />
        </w:r>
        <w:r>
                <w:fldChar w:fldCharType="end" />
        </w:r>
</w:p>
```

Looking at the previous example in detail:

1. The first r element contains the fldChar empty element, with its fldCharType attribute set to "begin" to indicate the start of a field command.
2. The next r element contains the instrText element, which contains the commands that tell Word how to process this field. To create a form text field, the instrText element must contain the text FORMTEXT.
3. The third r element holds another empty fldChar element with its fldCharType set to "separate". This marks the end of the field commands.
4. The fourth r element contains a t tag that holds the result of the field's processing (or the intial value for the field). In this example, five blank spaces have been used as the initial value for the field.
5. The final r tag holds the last fldChar element, this time with its fldCharType attribute set to "end" to mark the end of the definition.

For Word to process form fields correctly, the document must be protected for form editing only. You can turn on this level of protection by adding a documentProtection tag to the docPr element at the start of the WordML document. The edit attribute of the documentProtecton tag must be set to "forms" and the enforcement attribute must be set to "on". Here's an example:

```
<w:docPr>
        <w:documentProtection w:edit="forms" w:enforcement="on" />
</w:docPr>
```

After the field has been filled in by the user, the r element that contained the original value will hold the value entered by the user. The result would look like this if the user entered "My Data Entered" into the form field:

```
<w:p>
        <w:r>
                <w:fldChar w:fldCharType="begin"/>
        </w:r>
        <w:r>
        <w:instrText>FORMTEXT</w:instrText>
    </w:r>
    <w:r>
        <w:fldChar w:fldCharType="separate" />
    </w:r>
    <w:r>
        <aml:annotation aml:id="0" w:type="Word.Bookmark.Start"
            w:name="MyField" />
        <w:t>My Data Entered</w:t>
        <aml:annotation aml:id="0" w:type="Word.Bookmark.End" />
    </w:r>
    <w:r>
        <w:fldChar w:fldCharType="end" />
    </w:r>
</w:p>
```

Formatted: English (U.S.)

Formatted: English (U.S.)

Hyperlinks

A hyperlink has two components: the hyperlink itself (which the user will click on) and the target for the link. Potential targets include external files, e-mail addresses, and bookmarks. If you are creating a hyperlink in Microsoft Word, other targets are supported (e.g. the top of the document and headings). However, all of those targets are implemented by adding a bookmark at the appropriate location in the document. In this section, you'll see how to create a bookmark for a target within the document.

For a bookmark to be the target of a hyperlink, it must be a complete bookmark pair and have a name assigned to it. For instance, in Word if the user creates a hyperlink to the top of the document, a bookmark called "_top" is inserted at the top of the document. The resulting WordML looks like this:

```
<aml:annotation aml:id="0" w:type="Word.Bookmark.Start" w:name="_top" />
 <aml:annotation aml:id="0" w:type="Word.Bookmark.End" />
```

The hyperlink that points to this bookmark is represented in WordML by an hlink tag that has "_top" in its bookmark attribute. The text that is displayed by Word as the hyperlink must be inside a r tag between the hlink element's open and close tag (see Figure 10 for how the link appears in Word):

Deleted: 9

```
<w:hlink w:bookmark="_top">
    <w:r>
        <w:rPr>
            <w:rStyle w:val="Hyperlink" />
         </w:rPr>
             <w:t>Go To Top</w:t>
         </w:r>
</w:hlink>
```

**Insert graphic WordMLDev09.TIF

Figure 10: A hyperlink in Microsoft Word

Deleted: 9

You can use any style that you want with your hyperlink. However, the Hyperlink style that is generated by Microsoft Word is what most users will recognize as the visual clue for a hyperlink (underlined blue text). You should consider adding this style to your document and using it with your hyperlinks for consistency's sake:

[redacted]

[redacted]

Two other attributes of the hlink tag can be useful in generating a WordML hyperlink that will be read in Word:

- screentip: The text in this attribute is displayed when the user hovers their mouse over the hyperlink in Word (see Figure 10):

  ```
  <w:hlink w:bookmark="_top" w:screenTip="a screentip">
  ```

- nohistory: This attribute, when set to 'off', prevents the link from being added to the document's history list when the user clicks on it.

**Insert graphic WordMLDev10.TIF
Figure 10: A Microsoft Word Hyperlink with a screentip.

Formatted: Heading 2

Macros and Components

A document can also contain Visual Basic for Applications (VBA) code, toolbar modification, OLE/OCX components and other 'active' components. All of these items can be represented in WordML. In this document, you'll be introduced to how WordML stores VBA code and OCX/OLE components. You'll also see how Word ensures that software can detect whether these components are present in the document so that the component can, for instance, be scanned for viruses. Word also ensures that if components are not made visible in WordML then they will not be executed.

For VBA code, a base64 encoded version of the binary file generated by the VBA editor is held in the bindata element inside the docSuppData element. The binData element has a name attribute whose value must be set to "editdata.mso". The docSuppData element is a top-level element under the wordDocument root element and follows the styles element in a document created by Word.

Formatted: Indent: First line: 0.5"

A typical VBA module in a WordML document looks like this:

```
<w:docSuppData><w:binData w:name="editdata.mso">
QWN0aXZlTWltZQAAAfAEAAAA/////wAAB/AbDwAABA
```

...more binary data...
LgBNAFkATQBPAEQAVQBMAEUAAABAAAAL8AQAAAASNFZ4
</w:binData></w:docSuppData>

Representing an OLE/OCX element in WordML is more complicated than storing VBA code because an OLE/OCX component also has a graphical representation in the document. For OLE/OCX components, a binData element within a docOleData element is used to hold the OLE data. For OLE/OCX components, the name attribute of the binData element must be set to "oledata.mso".

```
<w:docOleData>
	<w:binData w:name="oledata.mso">
0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAA
...more binary data..
C4zcL+WTKDhJozVltEGRkTOwQAROjpejLDyT5d+/F5BeLt5n3wv4P/Cl4BK=
	</w:binData>
</w:docOleData>
```

Formatted: Indent: First line: 0.5"

Formatted: Indent: First line: 0.5"

Later in the document, a set of VML-related tags will handle the display of the component.

Two attributes on the wordDocument element are used to flag the presence of the VBA code and OCX/OLE components: macrosPresent for VBA code and embeddedObjectPresent for OCX/OLE components.

Formatted: Indent: First line: 0.5"

The macrosPresent attribute is used to indicate that macros are present in the document. If the attribute is missing or if it's set to "no" then Word won't load a document that has a docSuppData element. This attribute is strictly enforced. If, for instance, the attribute is present and is set to "yes" (indicating that macros are supposed to be present) and Word doesn't find a docSuppData element before it finds the body element, Word will refuse to load the document.

The second attribute is the embeddedObjectPresent attribute which indicates that an OCX or OLE component may have been used in the document. If the attribute is missing or if it's set to "no" then Word won't load a document that has a docOLEData element. This attribute is not, however, strictly enforced. If the attribute is present and is set to "yes" but Word doesn't find an docOLEData element before the body element, Word will still load the document.

Section 6: Auxiliary Tags

Included in a WordML document when that document is created by Microsoft Word are a number of tags that provide information to the application reading the document. These tags, from the [redacted]namespace, provide information about how Word handled the element. Setting these attributes and tags [redacted]. These tags are provided as a convenience to other WordML processing tools and provide a convenient way to access information that would otherwise be difficult to determine.

When creating a document, there is no problem with using the WordML sectPr tags while omitting the auxHint sect tags in your document. However, when reading a WordML document the sect elements provide containers for sections of your document. This can be very useful in processing the document, especially with XSLT which is oriented towards processing child elements inside container elements.

[redacted]

[redacted]

Formatted: Indent: First line: 0.5"

Deleted:

Sections and sub-sections

WordML does not use a container element for a section but, instead, marks the end of a section with a sectPr element. However, Word does generate sect tags to enclose the p tags that make up a section, creating a true XML container for sections. Within a sect tag, each table of contents heading generates sub-section tags that enclose content at a particular heading level or lower.

The sect Tag

The sect tags are relatively simple: a WordML document will consist of at least one sect element. Where the document contains multiple sectPr elements, defining multiple sections in the document, the document will consist of a series of sect tags. Including the sect tags in the definition of a WordML body element, this means that there are three possible structures for the body element:

- A single sect element:

```
<w:body>
        <wx:sect>
                <p>
                </p>
                ...etc. ...
        </wx:sect>
<w:body>
```

- Multiple sect elements:

```
<w:body>
        <wx:sect>
                <p>
                </p>
                ...etc. ...
        </wx:sect>
        <wx:sect>
```

```
                <p>
                </p>
                ...etc. ...
            </wx:sect>
            ... etc. ...
    <w:body>
```

- No sect elements (if the document was generated outside of Word)

```
<w:body>
        <p>
        </p>
        ...etc. ...
    <w:body>
```

The sub-section Tags

The sub-section tags are more complicated. A sub-section tag is generated whenever a paragraph is found that has an outlineLvl assigned in the p tag's pPr element. In this example, for instance, the paragraph is assigned to the second level of the outline (the lowest level is 0):

```
<w:p>
        <w:pPr>
                <w:outlineLvl w:val="2" />
        </w:pPr>
        <w:r>
                <w:t>x</w:t>
        </w:r>
</w:p>
```

Outline levels are frequently assigned through styles. In Word, for instance, the Heading 1 style has an outline level of 0 set in its rPr element. Any text formatted with the Heading 1 style picks up that outline level and generates a sub-section tag.

Deleted: will

Deleted: d

Word nests sub-sections within each other, depending on the outline level.When Word finds a paragraph with an outlineLvl assigned to it, Word generates a sub-section open tag. If the outlineLvl just found is higher than the previous outlineLvl, the new sub-section will be nested within the sub-section created for the earlier outlineLvel; if the previous outlineLvl was equal to or higher than the outlineLvl just found, close tags for all the higher level sub-section elementer are generated before the new sub-section element is opened.

In this example, for instance, there are five headings at various heading levels:

Heading Level 1

Paragraph1
Paragraph2

Heading Level 2

Paragraph3

Paragraph4

Heading Level 2
Paragraph5
Paragraph6

Heading Level 1
Paragraph7

Omitting all WordML tags, the auxiliary sect and sub-section tags that Word would generate would look like this:

```
<wx:sect>
        <wx:sub-section>
        Heading Level 1
        Paragraph1
        Paragraph2
                <wx:sub-section>
                Heading Level 2
                Paragraph3
                Paragraph4
                </wx:sub-section>
                <wx:sub-section>
                Heading Level 2
                Paragraph5
                Paragraph6
                </wx:sub-section>
        </wx:sub-section>
        <wx:sub-section>
        Heading Level 1
        Paragraph7
        </wx:sub-section>
</wx:sect>
```

Using sect and sub-section

Inserting a section break will create a new sect element in the document and close all open sub-section tags. In this sample, a section break has been added after paragraph4:

Heading Level 1
Paragraph1
Paragraph2

Heading Level 2
Paragraph3
Paragraph4
←Section Break→

Heading Level 1

Paragraph5

The resulting sect and sub-section tags would look like this:

```
<wx:sect>
        <wx:sub-section>
        Heading Level 1
        Paragraph1
        Paragraph2
                <wx:sub-section>
                Heading Level 2
                Paragraph3
                Paragraph4
                </wx:sub-section>
        </wx:sub-section>
</wx:sect>
<wx:sect>
        <wx:sub-section>
        Heading Level 1
        Paragraph5
        </wx:sub-section>
</wx:sect>
```

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

		[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

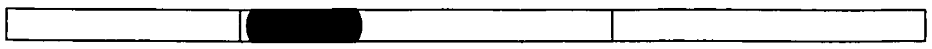
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]



[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]
[redacted]	
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
	[redacted]
[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]